k-vecinos más cercanos

Brenda Paola Quintana Silva

Julio 2019

1. Introducción

El algoritmo de los k vecinos más cercanos (k Nearest Neighbor) tiene como objetivo clasificar una observación en alguna categoría o grupo de acuerdo a datos previamente obtenidos. Usando una métrica (en el sentido matemático) se determinan los k puntos más cercanos (vecinos) y se etiqueta la nueva observación con la categoría o grupo de mayor frecuencia en el conjunto de puntos más cercanos.

E k vecino más cercano trabaja de la siguiente forma:

Sea X un espacio métrico, d una métrica sobre X, un conjunto de etiquetas $\{1, ..., M\}$ y un conjunto de n pares $\{(x_1, \theta_1), ..., (x_n, \theta_n)\}$, donde $x_i \in X$ y θ_i es la categoría de x_i.

Para clasificar un nuevo punto x, se tiene que estimar su etiqueta a partir de la muestra observada de x. El vecino más cercano para x es $\widehat{x}$ tal que $\widehat{x} \in argmin\{d(x, x_i)\},\ i = 1, ..., n$

Para clasificar a un nuevo punto la regla de los k vecinos más cercanos (k-NN) asigna a x la etiqueta que más se repita de entre los k vecino más cercanos como se menciona en [1].

2. Problema

Con el fin de diagnosticar enfermedades se crean protocolos que son un conjunto de procedimientos para determinar si un enfermo presenta un determinado cuadro clínico entonces tiene cierto padecimiento.

En este trabajo se quiere probar un protocolo médico y así determinar qué tan bueno es. Usando como herramienta el algoritmo k-NN y datos previamente observados de pacientes. La razón de esto es que es simple y aunque tiene muchas limitaciones, dará una idea del comportamiento de los pacientes en el aspecto de comparar si pacientes enfermos se mantienen cercanos de otros que también lo están, lo cual es una guía indirecta de posibles categorías del padecimiento investigado.

Se utilizará el conjunto de datos médicos de 44 pacientes y habrá sólo 2 etiquetas: 1 si el paciente está enfermo, 0 si no lo está. Se dirá que el protocolo

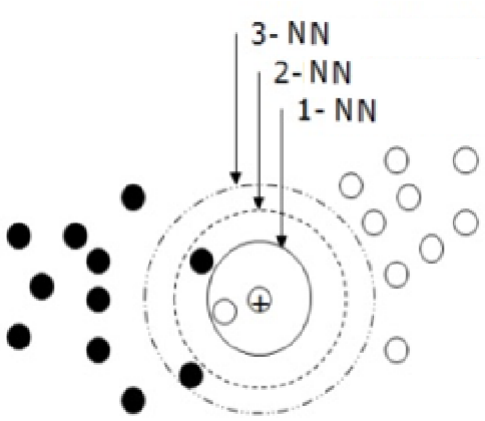


Figura 1: + se le asigna blanco bajo 1-NN, o bien + se asigna negro con 3-NN. En 2-NN no se puede dar una asignación

es bueno si con los datos que proporciona este se logra distinguir a los pacientes enfermos y de los sanos.

Los datos son mediciones para los cambias de posturas de diferentes variables.

En el Apéndice se muestra en R el código el cual se utilizó para analizar los datos.

Procedimiento: Organizar las variables para cada cambio de posición.

Sincope	Mean U	Peak U	EDV U	PI U	RI U	SD U	HR	tas	tad
1	42.45	69.86	28.16	0.97	0.59	2.49	68.81	90	60
1	41.32	63.49	27.64	0.85	0.56	2.29	68.23	112	70
1	52.9	71.33	38.1	0.62	0.46	1.87	87.59	90	60
1	36.33	58.61	23.64	0.94	0.58	NA	84.30	103.81	64.95
1	55.52	85.35	38.8	0.83	0.54	2.21	69.63	100	80
1	45.28	77.74	26.42	1.12	0.65	2.94	72.82	90	70
1	55.1	110.36	35.76	1.35	0.65	3.19	57.94	90	60
1	64.26	113.21	39.09	1.14	0.64	2.89	71.10	108.01	57.63
1	50.73	73.48	34.71	0.75	0.52	2.11	77.96	114	70
1	57.83	90.17	40.53	0.85	0.54	2.23	73.13	130	80
1	42.81	74.47	27.36	1.09	0.62	2.71	79.77	110	80
1	77.05	117.39	53.33	0.82	0.54	2.2	59.68	90	70
1	51.36	78.08	31.72	0.89	0.59	2.45	61.14	90	60
1	69.81	96.25	48.67	0.67	0.49	1.97	72.15	100	80
1	59.25	88.11	36.83	0.85	0.57	2.38	53.95	112	70
1	53.63	77.9	34.65	0.79	0.55	2.24	58.03	124	80

Figura 2: Valores de las diferentes variables en la posición Acostado1

Posteriormente se toma la diferencia de las variables y entre cada posición. $\{Sentado1 - Acostado1, Acostado2 - Sentado1, ...\}$ en total los 8 diferentes

cambios, como se ve en la figura 3.

Sincope	1	2	3	4	5	6	7	8	9
1	2.64	3.01	1.60	-0.01	-0.01	0.02	6.21	0.00	2.00
1	2.36	3.91	1.36	0.02	0.00	0.03	6.22	-2.00	2.00
1	-9.73	-9.80	-9.74	0.13	0.07	0.29	-14.45	0.00	2.00
1	-2.40	-6.81	-0.80	-0.10	-0.03	NA	15.42	7.36	3.29
1	-7.81	-5.96	-11.20	0.24	0.10	0.69	10.93	0.00	-10.00
1	-5.43	2.24	-5.71	0.35	0.08	0.92	9.39	0.00	-10.00
1	0.06	28.82	-1.03	0.52	0.08	0.84	14.90	2.00	0.00
1	-4.72	-8.18	-3.89	0.02	0.01	0.11	15.65	3.75	0.60
1	-4.71	-4.59	-5.02	0.09	0.04	0.23	1.31	-2.00	0.00
1	3.84	4.35	3.45	-0.04	-0.02	-0.07	4.13	-4.00	0.00
1	-5.77	-5.81	-4.95	0.15	0.04	0.39	9.76	2.00	0.00
1	-32.23	-39.67	-25.40	0.29	0.09	0.64	16.25	0.00	-10.00
1	-0.64	-0.48	-0.28	0.02	0.00	0.06	4.05	0.00	0.00
1	-3.27	-1.07	-4.45	0.09	0.04	0.20	6.15	-6.00	-10.00
1	-1.33	-0.86	-0.96	0.02	0.01	0.05	15.69	-2.00	0.00
1	0.55	0.08	1.63	-0.03	-0.02	-0.10	12.00	-12.00	2.00

Figura 3: Tabla de diferencias de Sentado1 - Acostado1

Posteriormente se parte en 2 el conjunto de datos inicial, uno de ellos será el conjunto de entrenamiento y el otro será el conjunto de prueba con el cual se probará el protocolo.

Se tomó el 80 % de los datos de la diferencia que será el conjunto de entrenamiento y el 20 % será el conjunto prueba.

Sincope	1	2	3	4	5	6	7	8	9
1	2.64	3.01	1.60	-0.01	-0.01	0.02	6.21	0.00	2.00
0	-7.53	-11.84	-2.58	-0.04	-0.02	-0.12	10.83	2.35	3.11
0	0.05	0.46	0.43	0.00	0.00	-0.01	4.52	0.07	0.84
0	-7.05	-9.71	-7.94	0.05	0.02	0.19	7.42	6.54	1.97
0	0.59	-11.21	-0.29	-0.16	-0.01	-0.22	4.11	0.74	3.54
0	-1.45	-1.81	-1.11	0.02	0.01	0.06	1.60	0.72	-0.25
0	-2.67	-4.22	-1.21	-0.02	-0.01	-0.05	1.55	0.02	-0.44
0	4.53	12.31	-1.37	0.09	0.03	0.64	1.74	9.02	4.64
0	1.46	3.71	-0.02	0.06	0.02	0.17	10.83	4.36	2.04
0	-9.50	-12.00	-7.36	0.05	0.02	0.09	8.05	5.84	3.26
0	-1.63	1.22	-5.54	0.25	0.07	2.32	1.75	-1.51	-0.30

Figura 4: Ejemplo de la tabla de prueba

Con todos los datos de conjunto prueba se les va a aplicar la regla del $k-NN$ y así se obtendrá una estimación de su etiqueta y se comparará con el resultado real.

Consideresé el caso donde d es la distancia euclidiana y $k = 1, 3$.

En figura 5 vemos la correcta ejecución del programa y asi en la siguiente figura vemos los resultados de un forma más comprima para $k = 1, 3$.

```
[1]   6  "1"
    Respuesta distancias orden permutacion
17          1   11.54372    17           1
21          0   15.71976    21           2
33          0   19.57762    33           3
[1]  16  "1"
    Respuesta distancias orden permutacion
17          1   12.78149    17           1
9           1   14.94061     9           2
40          0   15.47033    40           3
[1]  32  "0"
    Respuesta distancias orden permutacion
9           1   5.685904     9           1
40          0   9.081547    40           2
36          0  10.063732    36           3
[1]  39  "0"
    Respuesta distancias orden permutacion
28          0   5.655855    28           1
17          1  11.126828    17           2
33          0  13.190163    33           3
```

```
[1]   6  "1"
    Respuesta distancias orden permutacion
17          1   11.54372    17           1
[1]  16  "1"
    Respuesta distancias orden permutacion
17          1   12.78149    17           1
[1]  32  "0"
    Respuesta distancias orden permutacion
9           1   5.685904     9           1
[1]  39  "0"
    Respuesta distancias orden permutacion
28          0   5.655855    28           1
```

Figura 5: Ejecución del programa para $k = 1, 3$, la primera línea nos da el número del paciente y si tiene o no la enfermedad, también se muestra los 3 vecinos más cercano para ese paciente del lado izquierdo y del derecho se muestra el vecino más cercano

Como se puede ver en la figura 6 es muy variable en k = 1 las veces que acierta al predecir, en cambio para k = 3 se varianza es menor.

Figura 6: Ejecución del programa para $k = 3$, 34 elementos en la tabla de entrenamiento, 10 en la tabla de prueba

Para ser más preciso se saco la media de varias ejecuciones de este programa y para k = 3 el promedio de acierto que tiene es de 5.75 (57 % de la veces acierta) teniendo una tabla de entrenamiento de tamaño 34 y el tamaño de la tabla de prueba igual a 10.

Posteriormente se cambió el tamaño de tabla de entrenamiento para verificar si el gran tamaño del error provenía de los pocos datos que se tienen, ahora su tamaño será 40 y el tamaño de la tabla de prueba igual a 4, la k será siendo igual a 3 y así se obtuvo que su promedio fue de 2.41 (el 60.25 % de las veces acierta).

Para k =1 tomando el caso de 40 sujetos en la tabla de entrenamiento y 4 sujetos en la tabla de prueba se tiene como resultado que aproximadamente el 62.5 % de las veces acierta, por último sw tomó el caso de 34 sujetos en la tabla de entrenamiento y 10 sujetos en la tabla de prueba obteniendo que el 25 % acierta.

3. Conclusion

Como se observó en los resultados anteriores los datos arrojados por el algoritmo no son del todo confiables, siendo que el máximo porcentaje de predicciones acertadas es de al rededor del 60 %, un nivel bastante bajo.

Se estipula que este comportamiento es debido a la poca cantidad de datos disponibles para el correcto funcionamiento del algoritmo, ya que de acuerdo a lo mostrado anteriormente únicamente se usó un total de cuarenta y cuatro datos.

Se teoriza que teniendo una base de datos más amplia, mayor a ciento cincuenta, el algoritmo comenzaría a tener una mayor convergencia. De igual manera, esto se vería optimizado utilizando los tres vecinos más cercanos ($k = 3$) pues bajo esta condición se obtuvieron las mejores predicciones.

4. Apéndice

```
#distancia 1
dist1 <- function(x, y){
  return(sum(abs(x-y)))
}

#distancia Euclidania
distEuc <- function(x, y){
  return(sqrt(sum((x-y)**2)))
}

#Funcion que nos dice la diferencias entre posiciones
Diferencias <- function(tablaU){
  Ayuda  <- matrix(nrow=43,ncol=8, rep(0, 43*8,byrow=TRUE))
  indice <- 1

  for (i in 1:8) {
    Ayuda[,indice] <- c(as.numeric(tablaU[[3+i]]) - as.numeric(tablaU[[i+2]]))
```

```
    indice <- indice + 1
  }

  Ayuda <- as.data.frame(Ayuda)
  return(Ayuda)
}

#Ayuda a constrir tablas solo con un tipo de posicion
seleccion <- function(numero){
  lista <- c(1, 2, numero)
  for (i in 1:8) {
    numero <- numero +9
    lista[i+3] <- numero
  }
  return(lista)
}

#Genera un conjunto de 34 numero aleatorias es 1 -43
numAleatorios <- function(cantidad){
  numeros <- c(0)
  i <- 1
  while(length(numeros) != cantidad){
    numeros[i] <- ceiling(runif(1, min=0, max=44))
    if(length(numeros) == length(unique(numeros))){
      i <- i +1
    }
    numeros <- unique(numeros)
  }

return(sort(numeros))
}

#saca las distancias a un punto y las ordena de forma creciente
distEn11<- function(x, tablaU){
  ElConjunto <- vector('list', 34)
  Aux <- tablaU[, -c(1, 2)]

  for (i in 1:34) {
    xi <- as.numeric(Aux[i,])
    ElConjunto[[i]] <- xi
  }

  distancias <- c(0)
  orden <- c(1:length(ElConjunto))

  for (i in 1:length(ElConjunto)) {
```

```
    distancias[i] <- distEuc(x, ElConjunto[[i]])
  }

  Nombre <- as.vector(tablaU[, 1])
  Respuesta <-as.vector(tablaU[, 2])
  ayuda <- data.frame(Nombre, Respuesta, distancias, orden)
  sigma <- ayuda[with(ayuda, order(ayuda$distancias)), ]
  sigma$permutacion=c(1:length(ElConjunto))
  sigmaCool <- sigma[order(sigma$orden), ]
  return(sigmaCool)

}
#decide si esta o no enfermo para k = 3
identifica <- function(resultado){
  diagno <- 0
  if(sum(resultado) >= 2){
    diagno <- 1
  }
  return(diagno)
}

#funcion que compara el resultado obtenido del real y los cuenta
cuentaErrores <- function(res, diag){
  cuenta <- 0
  for (i in 1:length(res)) {
    if(res[i] == diag[i]){
      cuenta <- cuenta + 1
    }
  }
  return(cuenta)
}

#funcion para sacar la media
sacaMedia <- function(numInt, tamTabPrue){
  med <- 0

  for (i in 1:numInt) {
    elConjunto <- numAleatorios(44-tamTabPrue)
    tablaAleE <- Dif1[elConjunto,]
    tablaPrueba <- Dif1[-elConjunto,]

    nom <- c(0)
    res <- c(0)
    diag <- c(0)
    arreglo <- c(0)
```

```
    for (i in 1:tamTabPrue) {
      x <-tablaPrueba[i, 3:11]
      laTabl <- distEn11(x, tablaAleE)
      Tabla <- laTabl[order(laTabl$permutacion) ,]
      nom[i]<- tablaPrueba$ID[i]
      res[i] <- tablaPrueba$Sincope[i]
      arreglo[i] <-toString(Tabla$Respuesta[c(1:3)])
      diag[i] <- identifica( as.numeric( as.vector(Tabla$Respuesta[c(1:3)])) )
    }
    Resultados <- data.frame(nom, res, diag, arreglo)
    med <- cuentaErrores(Resultados$res, Resultados$diag) + med

  }
  return(med/numInt)
}
library("readr")
DatosPaulo <- read_csv(file.choose())
Pacientes <- DatosPaulo$PACIENTES



tablaAcostado1 <- DatosPaulo[, seleccion(5)]
nombre <- tablaAcostado1[1,]
tablaAcostado1 <- tablaAcostado1[-c(1),]
colnames(tablaAcostado1) <- nombre
MatrizAcostado1 <- matrix(nrow= 44,ncol =
                  9,as.numeric(as.matrix(tablaAcostado1[,-c(1,2)])))

tablaSentado1 <- DatosPaulo[, seleccion(6)]
tablaSentado1 <- tablaSentado1[-c(1),]
colnames(tablaSentado1) <- nombre
MatrizSentado1 <- matrix(nrow= 44,ncol =
                 9,as.numeric(as.matrix(tablaSentado1[,-c(1,2)])))

dif1 <- MatrizSentado1 -MatrizAcostado1
Dif1 <- cbind(tablaAcostado1[,1],tablaAcostado1[,2], dif1 )

tablaAcostado2 <- DatosPaulo[, seleccion(7)]
tablaAcostado2 <- tablaAcostado2[-c(1),]
colnames(tablaAcostado2) <- nombre
MatrizAcostado2 <- matrix(nrow= 44,ncol =
                  9,as.numeric(as.matrix(tablaAcostado2[,-c(1,2)])))

dif2 <- MatrizAcostado2 -MatrizSentado1
Dif2 <- cbind(tablaAcostado1[,1],tablaAcostado1[,2], dif2 )
```

```
tablaSentado2 <- DatosPaulo[, seleccion(8)]
tablaSentado2 <- tablaSentado2[-c(1),]
colnames(tablaSentado2) <- nombre
MatrizSentado2 <- matrix(nrow= 44,ncol =
                9,as.numeric(as.matrix(tablaSentado2[,-c(1,2)])))

dif3 <- MatrizSentado2 - MatrizAcostado2

tablaAcostado3 <- DatosPaulo[, seleccion(9)]
tablaAcostado3 <- tablaAcostado3[-c(1),]
colnames(tablaAcostado3) <- nombre
MatrizAcostado3 <- matrix(nrow= 44,ncol =
                 9,as.numeric(as.matrix(tablaAcostado3[,-c(1,2)])))

dif4 <- MatrizAcostado3 - MatrizSentado2

tablaDePie <- DatosPaulo[, seleccion(10)]
tablaDePie <- tablaDePie[-c(1), ]
colnames(tablaDePie) <- nombre
MatrizDePie <- matrix(nrow= 44,ncol =
              9,as.numeric(as.matrix(tablaDePie[,-c(1,2)])))

dif5 <- MatrizDePie - MatrizAcostado3

tablaAcostado4 <- DatosPaulo[, seleccion(11)]
tablaAcostado4 <- tablaAcostado4[-c(1), ]
colnames(tablaAcostado4) <- nombre
MatrizAcostado4 <- matrix(nrow= 44,ncol =
                9,as.numeric(as.matrix(tablaAcostado4[,-c(1,2)])))

dif6 <- MatrizAcostado4 - MatrizDePie

tablaEsfuerzo <- DatosPaulo[, seleccion(12)]
tablaEsfuerzo <- tablaEsfuerzo[-c(1), ]
colnames(tablaEsfuerzo) <- nombre
MatrizEsfuerzo <- matrix(nrow= 44,ncol =
                9,as.numeric(as.matrix(tablaEsfuerzo[,-c(1,2)])))

dif7 <- MatrizEsfuerzo - MatrizAcostado4

tablaAcostado5 <- DatosPaulo[, seleccion(13)]
tablaAcostado5 <- tablaAcostado5[-c(1), ]
colnames(tablaAcostado5) <- nombre
MatrizAcostado5 <- matrix(nrow= 44,ncol =
                9,as.numeric(as.matrix(tablaAcostado5[,-c(1,2)])))
```

```
dif8 <- MatrizAcostado5 - MatrizEsfuerzo

##########################################

elConjunto <- numAleatorios(40)
tablaAleE <- Dif1[elConjunto,]
tablaPrueba <- Dif1[-elConjunto,]

#Algortmo para k = 3
nom <- c(0)
res <- c(0)
diag <- c(0)
arreglo <- c(0)

#primera tabla de resultados
for (i in 1:4) {
  x <-tablaPrueba[i, 3:11]
  print(c(tablaPrueba$ID[i], tablaPrueba$Sincope[i] ))
  laTabl <- distEn11(x, tablaAleE)
  Tabla <- laTabl[order(laTabl$permutacion) ,]
  print(Tabla[c(1),])
}


#tabla resumida de resltados
for (i in 1:4) {
  x <-tablaPrueba[i, 3:11]
  laTabl <- distEn11(x, tablaAleE)
  Tabla <- laTabl[order(laTabl$permutacion) ,]
  nom[i]<- tablaPrueba$ID[i]
  res[i] <- tablaPrueba$Sincope[i]
  arreglo[i] <-toString(Tabla$Respuesta[c(1:3)])
  diag[i] <- identifica( as.numeric( as.vector(Tabla$Respuesta[c(1:3)])) )
}
Resultados <- data.frame(nom, res, diag, arreglo)
cuentaErrores(Resultados$res, Resultados$diag)

sacaMedia(12, 10)
```

Referencias

[1] S. Ben-David S. Shalev-Shwartz. *Understanding Machine Learning: From Theory to Algorithms*. EUA, Cambridge University Press, (2014).

Recontrucción de gráficas con estructura circuclar

Brenda Paola Quintana Silva

Julio 2019

1. Gráficas circulantes

Sea $G = (V, E)$ una gráfica simple no dirigida. Sea $f : V \to \{1, ..., |V|\}$ una función de etiquetado. Se define el peso de las aristas como el valor absoluto de la diferencia entre las etiquetas asignadas a u y v. El problema del "arreglo lineal mínimo"(layout) consiste en encontrar un etiquetado de los vértices tal que la suma de los pesos de las aristas sea mínima, es decir, es encontrar una función f tal que la siguiente suma sea mínima:

$$\sum_{u,v \in E} |f(u) - f(v)|$$

Este problema es categorizado como *NP-hard*. En el trabajo "Layout of random circulant graph de Richter y Rocha, se da una solución de orden polinomial para este problema bajo el supuesto de tener inicialmente una gráfica circulante. Este método ayuda a dar una solución al problema de la reconstrucción de secuencias del ADN, también es usado para el modelado de la corteza cerebral [1].

Una gráfica circulante H se debe definir sobre el conjunto de vértices $\{1, ..., n\}$ y sobre el conjunto de aristas dado por:

$E = \{(i, j) : |i - j| \equiv s(m\acute{o}d\ n), s \in S\}, \quad S \subseteq \{1, ..., n\}, \quad -s \equiv s(m\acute{o}d\ n)$

En el trabajo de Richter y Rocha se tiene una diferente definición de gráficas circulantes que no cumple con que su matriz de adyacencia también sea circulante. Bajo la condiciones de dicho estudio H se define para el conjunto de vertices $V = \{1, ..., n\}$ y de aristas $E = \{(i, j) : |i - j| \equiv s(\ \text{mód}\ n), s \in S\}$ donde $S \subseteq \{1, ..., \frac{n-1}{2}\}$. Con esta definición los resultados son diferentes pues la matriz que se obtiene es simétrica más no circulante.

En las Figuras 1, 2 y 3 se muestran algunos ejemplos típicos de gráficas circulantes.

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]
[1,]	0	0	1	0	1	0	1	0
[2,]	0	0	0	1	0	1	0	1
[3,]	1	0	0	0	1	0	1	0
[4,]	0	1	0	0	0	1	0	1
[5,]	1	0	1	0	0	0	1	0
[6,]	0	1	0	1	0	0	0	1
[7,]	1	0	1	0	1	0	0	0
[8,]	0	1	0	1	0	1	0	0

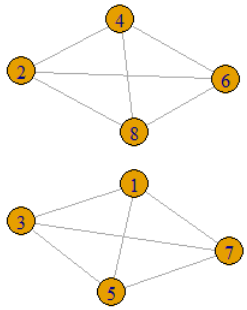
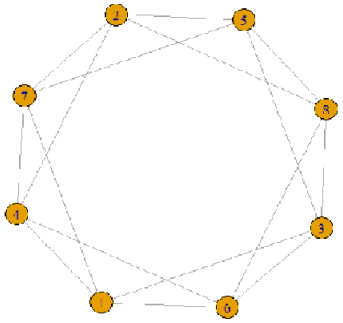
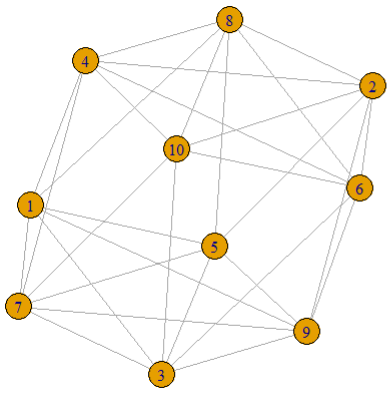


Figura 1: $n = 8$, $-s = \{\text{-2, -4, -6}\}$, $s = \{2, 4, 6\}$



	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]
[1,]	0	0	1	1	0	1	1	0
[2,]	0	0	0	1	1	0	1	1
[3,]	1	0	0	0	1	1	0	1
[4,]	1	1	0	0	0	1	1	0
[5,]	0	1	1	0	0	0	1	1
[6,]	1	0	1	1	0	0	0	1
[7,]	1	1	0	1	1	0	0	0
[8,]	0	1	1	0	1	1	0	0

Figura 2: $n = 10$, $-s = \{-5, -3, -2, -4\}, s = \{5, 3, 2, 4\}$



	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
[1,]	0	0	1	1	1	0	1	1	1	0
[2,]	0	0	0	1	1	1	0	1	1	1
[3,]	1	0	0	0	1	1	1	0	1	1
[4,]	1	1	0	0	0	1	1	1	0	1
[5,]	1	1	1	0	0	0	1	1	1	0
[6,]	0	1	1	1	0	0	0	1	1	1
[7,]	1	0	1	1	1	0	0	0	1	1
[8,]	1	1	0	1	1	1	0	0	0	1
[9,]	1	1	1	0	1	1	1	0	0	0
[10,]	0	1	1	1	0	1	1	1	0	0

Figura 3: $n = 10$, $-s = \{-2, -4, -7, -8, -6, -3\}, s = \{2, 4, 7, 8, 6, 3\}$

Una gráfica circulante aleatoria es el resultado de borrar aristas de aleatoriamente de la estructura de H con una probabilidad de $1-p$ donde $p \in [0,1]$.

Con la ayuda de los eigenvectores de la matriz de adyacencia es posible proponer un método para recuperar un *layout* (diseño) de una gráfica con estructura circulante.

2. Método

Una matriz circulante A se define con un solo vector $a \in R^n$, donde cada renglón de la matriz es una permutación del anterior. Así, A se ve como:

$$A = \begin{bmatrix} a_1 & a_2 & a_3 & \cdots & a_n \\ a_n & a_1 & a_2 & \cdots & a_{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_2 & a_3 & a_4 & \cdots & a_1 \end{bmatrix}$$

La matriz de adyacencia A de una gráfica circulante H, es una matriz circulante también. La gráfica aleatoria a considerar es $G = (V, E)$ que es resultado de borrar aristas con una probabilidad $1-p$ de H.

Por otra parte, si se define $M = pA$, donde M y A son matrices circulantes, M necesariamente describe la estructura de H. Por ejemplo:

$$M = p \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Sea $\widehat{M}$ la matriz de adyacencia de la gráfica aleatoria G, y las entradas de $\widehat{M}$ corresponden a variables aleatorias independientes Bernoulli, donde $P(\widehat{m_{ij}} = 1) = m_{ij}$, y donde a su vez m_{ij} son las entradas de M.

El etiquetado de G generalmente no se tiene, y únicamente se asume que esta gráfica viene de una gráfica circulante. Por esta razón, se necesita encontrar un embebimiento para G que tenga el orden correcto (un *layout*).

A continuación, se presenta un algoritmo que resuelve este problema usando los eigenvectores correspondientes al 2^{do} y 3^{er} eigenvalor más grande de $\widehat{M}$.

Algoritmo

Sea $\widehat{M}$ una matriz aleatoria:

1. Obtener los eigenvectores $\widehat{x}, \widehat{y}$ asociados a los eigenvalores $\lambda_2\left(\widehat{M}\right)$ y $\lambda_3(\widehat{M})$.
2. Obtener la coordenada angular φ_i para el punto $(\widehat{x_i}, \widehat{y_i})$.
3. Definir la permutación σ tal que $\sigma(i) > \sigma(j)$, si $\varphi_i \geq \varphi_j$.
4. Devolver la permutación σ.

En el Apéndice se muestra el código escrito en lenguaje R utilizado para la implematación de este algoritmo.

Pruebas del algoritmo

A continuación se muestran los resultados arrojados por el programa computacional usando el algoritmo previamente mencionado. Estas figuras fueron obtenidas utilizando matrices simétricas no circulantes.

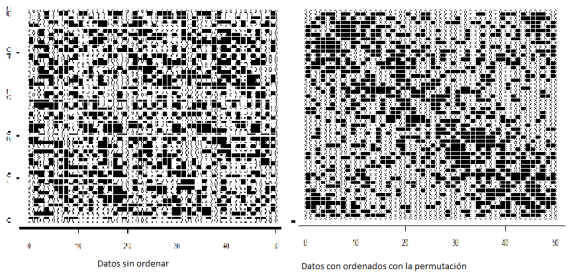


Figura 4: Reordenamiento de una matriz aleatoria simétrica (no circulante) de tamaño $50x50$.

Primeramente, En la Figura 4 se muestra como a partir de una matriz aleatoria simétrica se reorganizan todos los vertices para intentar obtener una forma circulante.

En la Figura 5 se aprecia la importacia que tiene en el procedimeinto el hecho de que la matriz inicial sea circulante, y no solos simétrica, para poder formar el círculo unitario con los eigenvectores de M.

El fenómeno de que los eigenvectores por coordenadas formen un círculo puede explicarse por la naturaleza de las matrices circulantes.

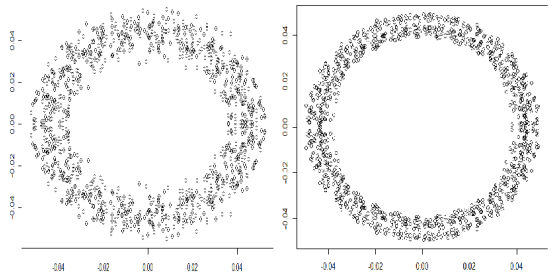


Figura 5: Gráfica de las coordenadas del 2^{do} y 3^{er} eigenvector de la matriz M.

3. Análisis de resultados

El programa se ejecutó varias veces para verificar que el algoritmo tuviera la eficiencia y eficacia que se menciona, sin embargo se puede observar en las siguientes ejecuciones que el error mencionado en [1] es muy grande.

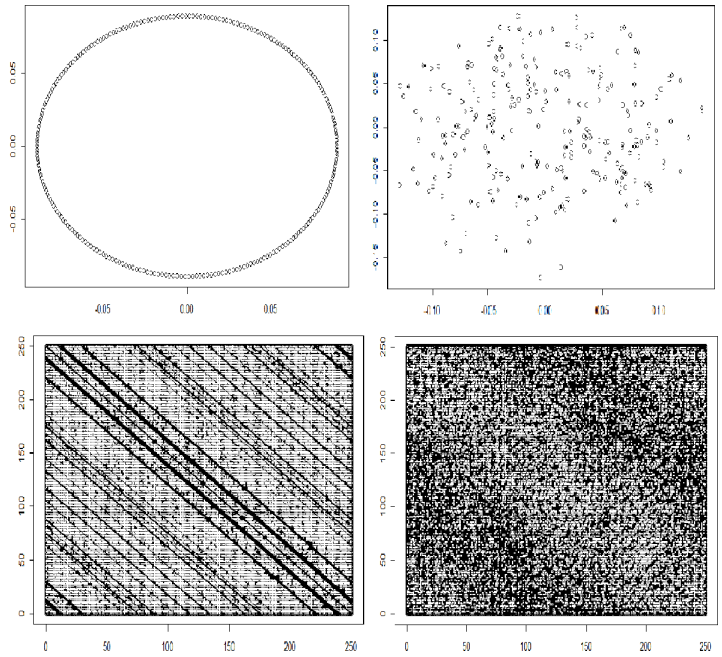


Figura 6: La primera gráfica correponde a la gráfica coordenada a coordenada del 2^{do} y 3^{er} eigenvector de M. La segunda corresponde a los eigenvectores de su matriz aleatoria. Se muestra el reordenamiento de 250 vértices, $|s| = 150$, $p = 0.5$.

En la Figura 6 puede observarse un ligero reordenamiento similar al que se menciona en [1], más no es el esperado. Además, la gráfica del 2^{do} y ^{3er} eigenvector de la matriz aleatoria no tiene la forma circular esperada.

Es así que se puede decir que a partir de un conjunto $|s|$=150 con probabilidad p=0.5, no se pudo reordenar completamente de la gráfica de 250 vertices.

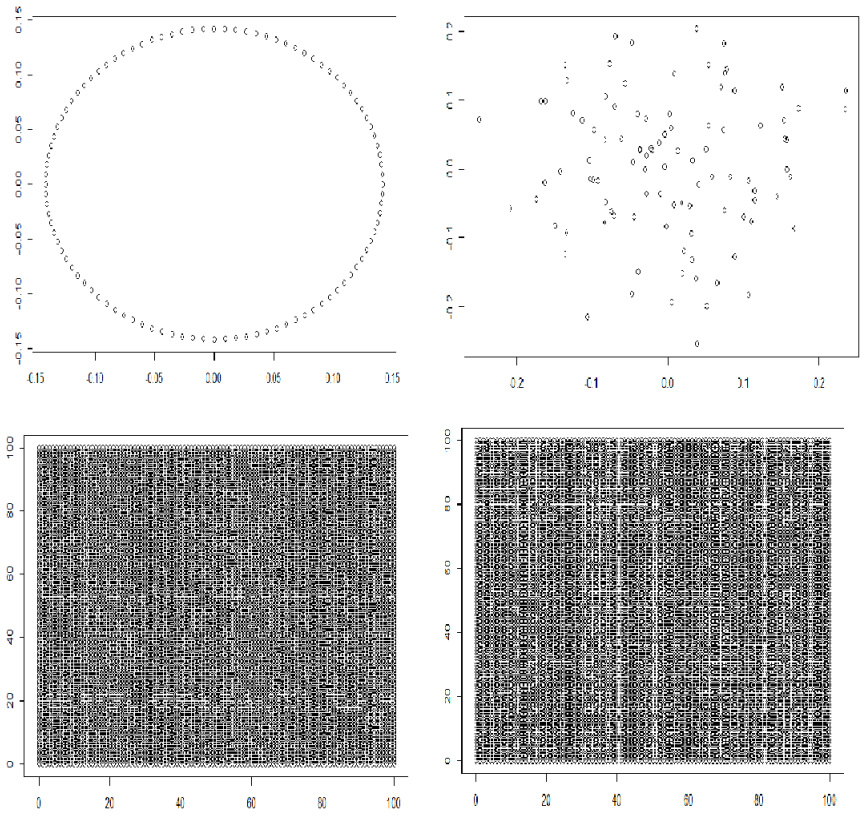


Figura 7: La primera gráfica correponde a la gráfica coordenada a coordenada del 2^{do} y 3^{er} eigenvector de M. La segunda corresponde a los eigenvectores de $\widehat{M}$. Se muestra el reordenamiento de 250 vértices, $|s| = 153$, $p = 0.8$.

Para la simulación hecha bajo las condiciones mostradas en la Figura 7 que resaltar es que la reordenación no es muy eficente ya que en esta misma figura no hay cambios significativos entre ambas gráficas. Esto podría deberse a que se tiene una gráfica casi completa, ya que la probabilidad de remover un arista era muy baja.

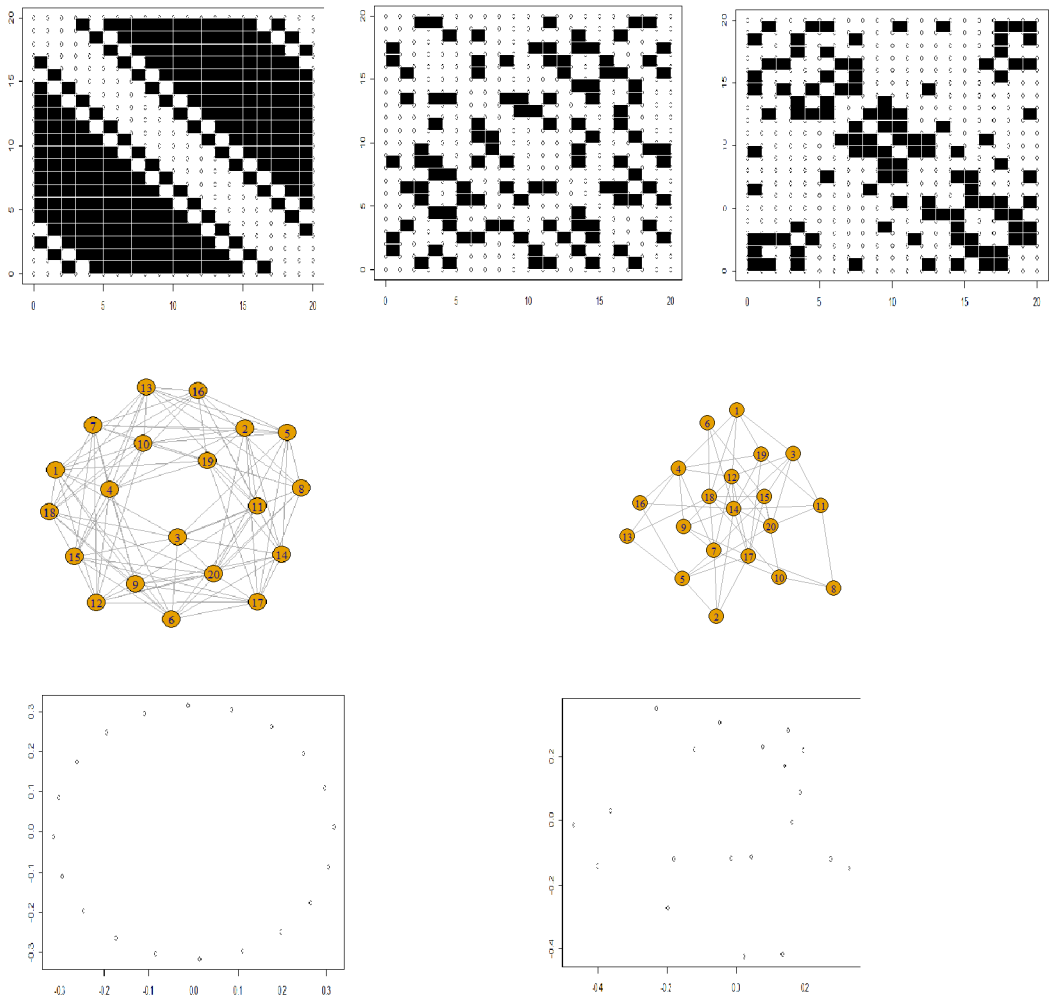


Figura 8: Diagrama de las matrices M, $\widehat{M}$ y el reordenamiento de $\widehat{M}$. También se dibujan sus gráficas y las gráficas coordenada a coordenada del 2^{do} y 3^{er} eigenvector de M y $\widehat{M}$ de una gráfica de 20 vértices, $|s| = 18$, $p = 0.5$

En el ejemplo de la Figura 8 se muestra la gráfica circulante y la gráfica aleatoria asociadas a la matrices M y $\widehat{M}$, respectivamente. También se anexan las gráficas del 2^{do} y 3^{er} eigenvector correspondientes y el rerodenamiento.

En este caso se obtiene el reordemnaineto esperado, se puede estipular que si tienen pocas aristas se puede hacer un mejor layout.

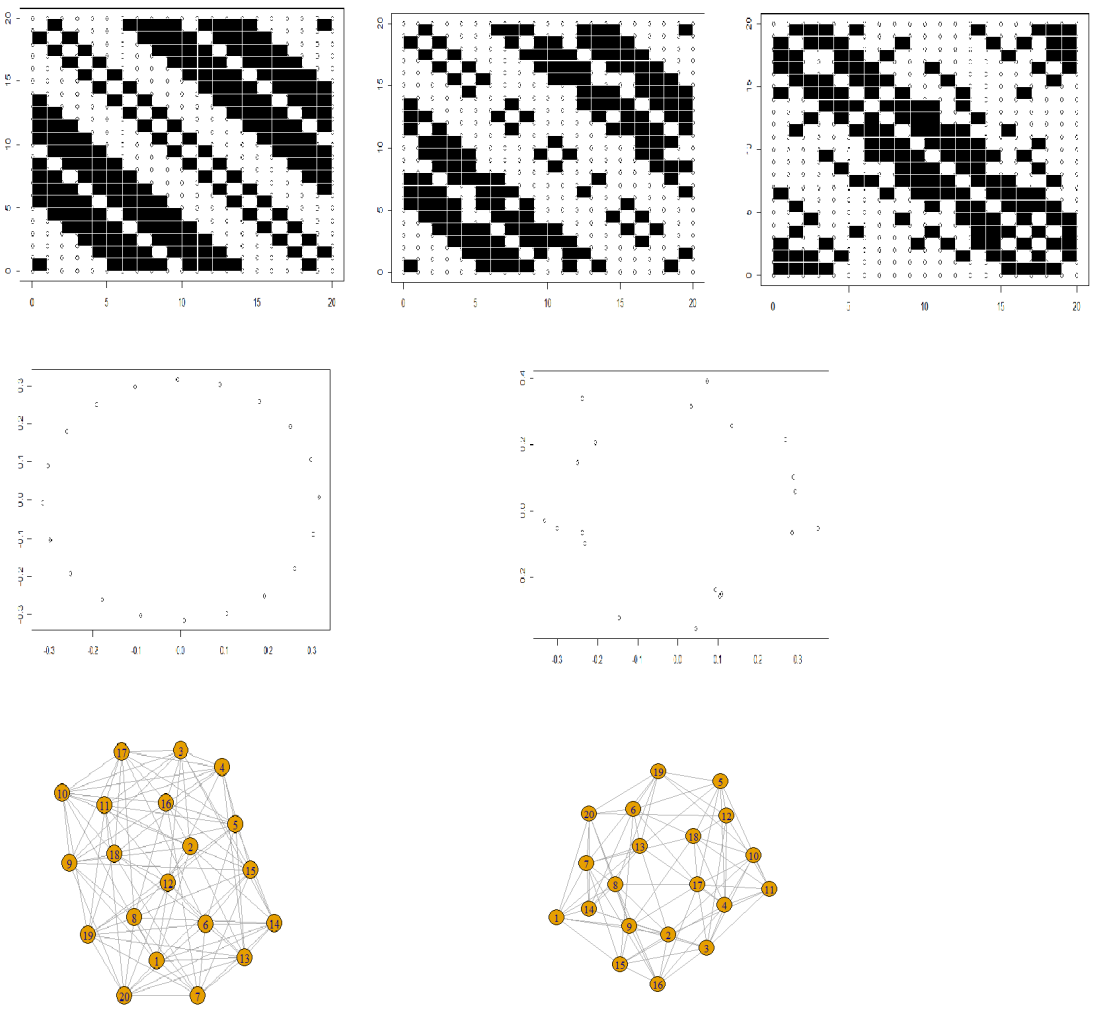


Figura 9: Diagrama de las matrices M, $\widehat{M}$ y el reordenamiento de $\widehat{M}$. Tambien se dibuja sus gráficas y las gráficas coordenada a coordenada del 2^{do} y 3^{er} eigenvector de M y $\widehat{M}$, de una gráfica de 20 vértices, $|s| = 10$, $p = 0.8$

En la simulación mostrada la Figura 9 se obtuvo un reordenamiento exitoso tomando una probabiliadad alta. Esto es indiciio de que el error en el reordenamiento provine de usar gráficas con muchas aristas.

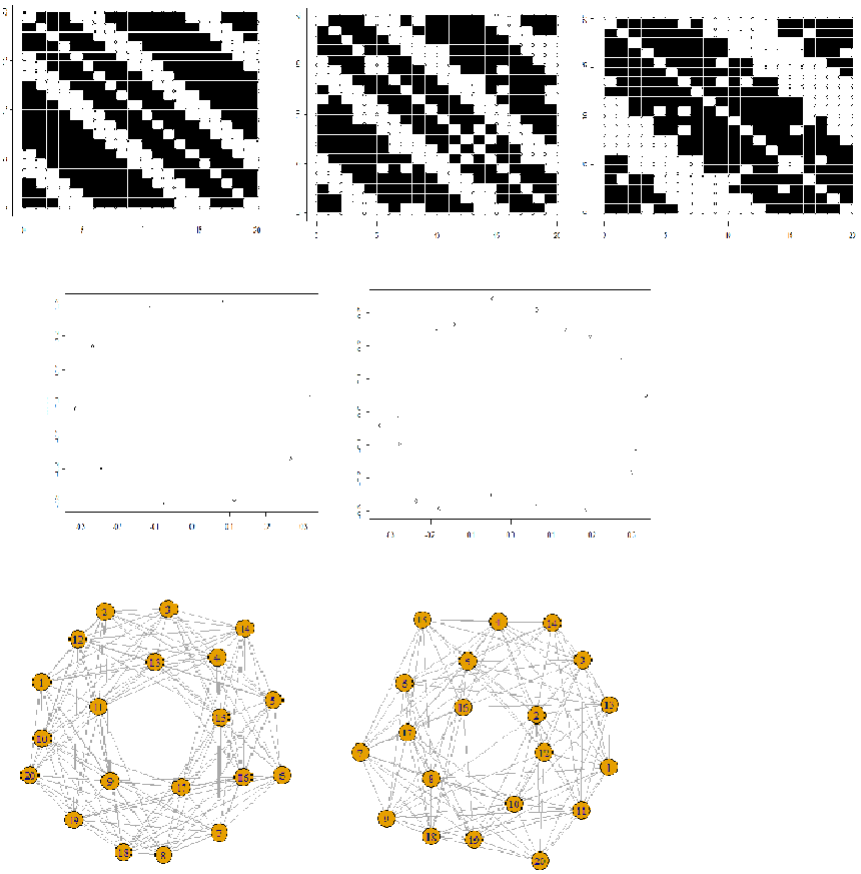


Figura 10: Diagrama de las matrices M, $\widehat{M}$ y el reordenamiento de $\widehat{M}$. Tambien se dibuja sus gráficas y las gráficas coordenada del 2^{do} y 3^{er} eigenvector de M y $\widehat{M}$, de una gráfica de 20 vértices, $|s| = 13$, p= 0.9

La simmulación ejemplificada en la Figura 10 es una muy buena ejecución de lo esperado. Esto pues los puntos coordenada a coordenada del 2^{do} y 3^{er} eigenvector de $\widehat{M}$ están muy cercanos a circulo unitario, por lo que el reordenaimeto que se obtuvo es muy bueno.

Los resultados mostrados en la Figura 11 se compilaron partiendo de una

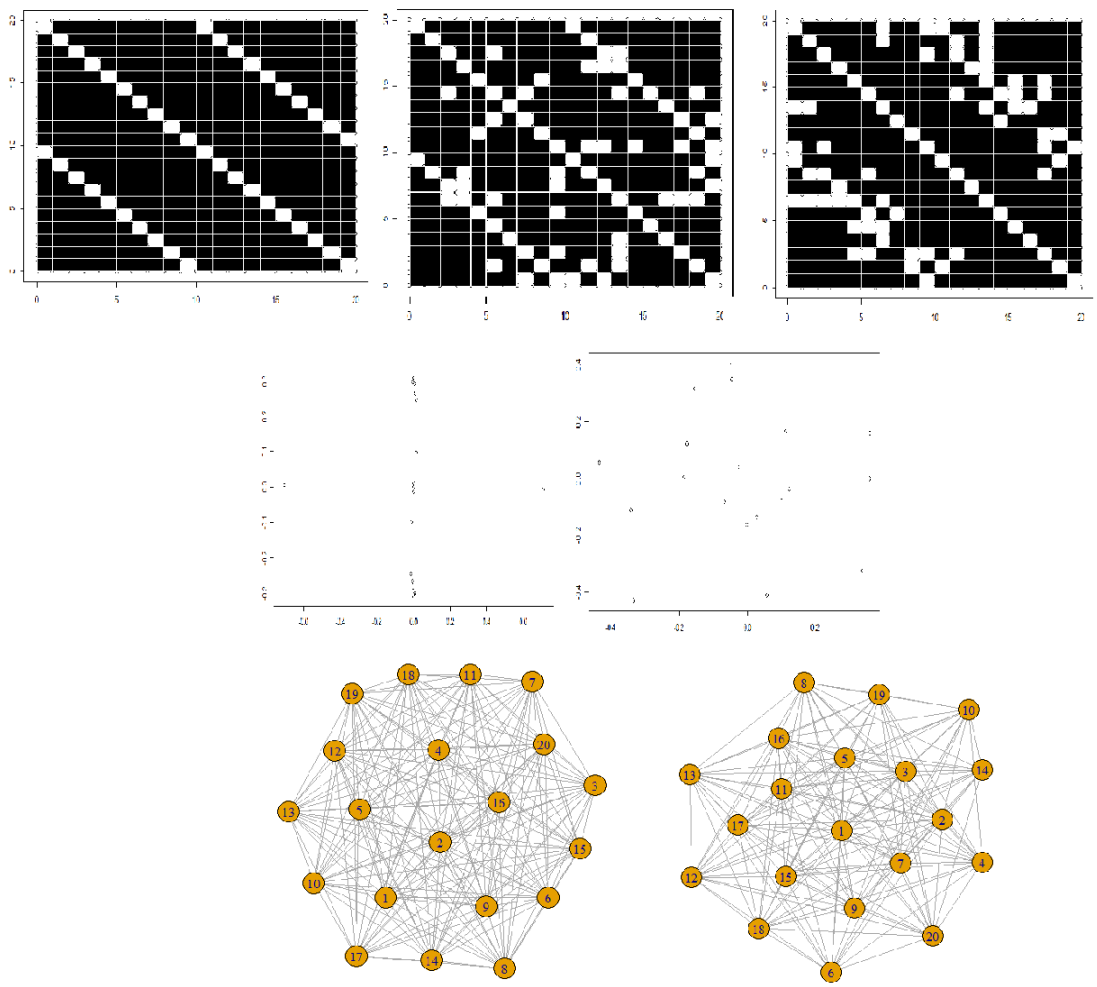


Figura 11: Diagrama de las matrices M, $\widehat{M}$ y el reordenamiento de $\widehat{M}$. Tambien se dibuja sus gráficas y las gráficas coordenada del 2^{do} y 3^{er} eigenvector de M y $\widehat{M}$, de una gráfica de 20 vertices, $|s| = 18$, $o = 0.9$

gráfica casi completa y circulante. Esto afecta demasiado a los eigenvectores pues ahora en ninguno de los casos las coordenadas del 2^{do} y 3^{er} eigenvalor se encuentran sobre la circufenrencia unitaria.

4. Conclusión

En primera instancia se concluye que el algoritmo no tiene un buen desempeño en algunos ejemplos que se simularon como los mostrados en la Figura 7, donde la gráfica es casi completa, el número de vértices muy grande y una probabilidad cercana a uno.

La propiedad de que (x_i, y_i), donde (x_i) e (y_i) son las entradas de los eigenvectores correspondientes al 2^{do} y 3^{er} eigenvalor más grande, se encuentren cerca de la circunferencia unitaria no se observa en todos las simulaciones. Estos casos que se deben analizar en trabajos posteriores.

5. Apéndice

```
library("igraph")

#funcion que dibuja el grafo
Dibuja <- function(A){
  plot(graph.adjacency(A, mode = "undirected"))
}

#Funcion que hace la matriz aleatoria
matrizAle <- function(Mat){
  gorro <- matrix(nrow = sqrt(length(Mat)),
           ncol =sqrt(length(Mat)), rep(0, length(A)))

  for (i in 1:(sqrt(length(Mat))-1)){
    for (j in (1+i):sqrt(length(Mat))) {
      gorro[i, j] <- rbinom(1, 1, Mat[i,j])
      gorro[j, i] <- gorro[i, j]
    }
  }
  return(gorro)
}

#funci n que saca los valores y vectores propios
propios <- function(matriz){
  valores <- eigen(matriz)$values
  vectores <- eigen(matriz)$vectors

  segVal <- valores[2]
  terVal <- valores[3]
```

```
  segVect <- vectores[,2]
  terVect <- vectores[,3]

  plot(segVect, terVect, type = "p")
  return(list(segVect, terVect))
}

#Fn que convierte los angulos del arctan en algulos positivos
convTan <- function(x, y){
  if(x >= 0 && y > 0){
    theta <- atan(y/x)
  }else if (y >= 0 && x < 0){
    theta <- pi + atan(y/x)
  }else if (x <= 0 && y < 0 ){
    theta <- atan(y/x) + pi
  }else{
    theta <- 2*pi + atan(y/x)
  }
  return(theta)
}

calculaPermutacion <- function(vectores){
  angulos <- c(0)
  segVect <- vectores[[1]]
  terVect <- vectores[[2]]
  for (i in 1:length(segVect)) {
    angulos[i] <- convTan(segVect[i], terVect[i])
  }
  orden <- c(1:length(angulos))
  ayuda <- data.frame(angulos, orden)
  sigma <- ayuda[with(ayuda, order(-ayuda$angulos)), ]
  sigma$permutacion=seq(from = length(angulos), to = 1, by = -1)
  sigmaCool <- sigma[order(sigma$orden), ]

  return(sigmaCool)
}

#funcion que hace la matriz de permutaci n de acuerdo al
 dateframe de la permutacion
matPermuta <- function(A, sigma){
  matPer <-  matrix(nrow=sqrt(length(A)),ncol=sqrt(length(A)),
               rep(0, sqrt(length(A))),byrow=TRUE)
  for (i in 1:sqrt(length(A))) {
    for (j in 1:sqrt(length(A))) {
      matPer[sigma$permutacion[i], sigma$permutacion[j]] <- A[i, j]
    }
```

```
  }

  return(matPer)
}
#Funcion que hace un diagrama de cuadros de acuerdo a una matriz
diagrama <- function(A) {
  base <- function(n, m){
    x <- c(0, 0, 1, 1)+n
    y <- c(0, 1, 1, 0)+m
    polygon(x, y, col = "black", border = "white")
  }

  xs <- c(0, 1:sqrt(length(A)))
  ys <- c(0, 1:sqrt(length(A)))
  cartesiano<- expand.grid(xs, ys)
  plot(cartesiano$Var1, cartesiano$Var2)

  for(i in 1:sqrt(length(A))){
    for(j in 1:sqrt(length(A)) ){
      if(A[i, j] == 1){
        base(j-1, sqrt(length(A))-i)
      }
    }
  }

}
#codigo a ejecutar

 n = 20 #numero de vertices a poner
 m= 7 #un tama o aproximado de s
 p = 0.8 #probilidad con la que se retiran las aristas

 numeros <- ceiling(runif(m, min=1, max=(n-1)))
 vertices <- c(1:n)
 grafica <- matrix(nrow = length(vertices),
         ncol =length(vertices), rep(0, length(vertices)**2))
 SYMenosS <- FacilS(n, numeros)
 s <- SYMenosS$s

 for (k in 1:length(s)) {
   for (i in 1:n) {
     for (j in 1:n) {
       if((( abs(i-j)) %%n) == s[k]){
         grafica[i, j] <- 1
       }
     }
```

```
  }
}
A <- grafica
M <- p*A
gorroMcool <- matrizAle(M)
a<- propios(M)

vectores1 <- propios(gorroMcool)
sigma <-  calculaPermutacion(vectores1)

diagrama(gorroMcool)
matPer <- matPermuta(gorroMcool, sigma)
diagrama(matPer)
```

Referencias

[1] Sebastian Richter e Israel Rocha. *Layout of random circulant graphs.* 2017.